SPECIFICATION

Electronic Version 1.2.8

Stylesheet Version 1.0

Method for Content-Aware Redirection and Content Renaming

Background of Invention

[0001] The present invention relates to content distribution in packet-switched networks.

[0002] Packet-switched networks, such as networks based on the TCP/IP protocol suite, can be utilized to distribute a rich array of digital content to a variety of different client applications. The most popular applications on the Internet today are browsing applications for searching the World Wide Web, e.g. Netscape Navigator or Microsoft Internet Explorer, which utilize the HyperText Transfer Protocol (HTTP) to retrieve documents written in the HyperText Markup Language (HTML) along with embedded content. See, e.g., R. Fielding et al., "Hypertext Transfer Protocol -- HTTP/1.1," IETF Network Working Group, RFC 2616 (1999), which is incorporated by reference herein. Other protocols for delivering data such as streaming media across the Internet include the Real Time Streaming Protocol (RTSP). See, e.g., H. Schulzrinne et al., "Real Time Streaming Protocol (RTSP)," IETF Network Working Group, RFC 2326 (April 1998), which is incorporated by reference herein. Resources on the Internet, such as HTML documents or multimedia content, are identified by Uniform Resource Identifiers (URIs). See, e.g., T. Berners-Lee et al., "Uniform Resource Identifiers (URI): Generic Syntax," IETF Network Working Group, RFC 2396 (August 1998), which is incorporated by reference herein. URIs can be expressed by a representation of their location-dependent network access mechanism, i.e. as a Uniform Resource Locator (URL) (e.g. "http://www.xyz.com/dir/document.html"), or by a persistent name referred to as a Uniform Resource Name (URN).

[0003] It is often advantageous when distributing content across a packet-switched

network to divide the duty of answering content requests among a plurality of geographically dispersed servers. Companies such as Akamai Technologies, Digital Island, AT&T and Adero provide services - referred to in the art as "content distribution" services - utilizing architectures which dynamically redirect content requests to a cache advantageously situated closer to the client issuing the request. Such network architectures are referred to herein generically as "content distribution networks" or "CDNs" for short. In its simplest form, content distribution networks consist of origin servers and edge servers. Clients connect to edge servers to request content. Requested content may already be in the edge server that the client connect to (for example if all of the edges are pre-populated with all of the content), or the edge server in question might fetch the content from the origin server on-demand if it does not already have the requested content. These two extremes, namely complete pre-population of the edges and on-demand loading of the edges, is clearly suboptimal in terms of storage and latency respectively. This is particularly true in the case of high quality streaming content where storing all of the content in all of the edge servers will not be feasible. At the same time if say a particular large piece of digital content, e.g. a movie, is already at an edge server, a new client requesting the same movie might potentially be best served from the same edge server.

Summary of Invention

[0004] The present invention is directed to mechanisms for content-aware redirection and content exchange/content discovery that permit a request for content to be redirected to a particular advantageous server that can serve the content. In accordance with an embodiment of the invention, the content-aware redirection mechanism roughly consists of a front-end and a back-end. The front-end makes use of the requested content to redirect a request to an edge server where the content might already be present or an edge that should request the content from the origin. The back-end takes part in a content exchange/content discovery process to determine where content is currently located. The requested content can advantageously be rewritten during the redirection process to not require the name of the requested and served object to be the same. Content renaming can be utilized for example to efficiently map different content names to the same piece of stored content.

[0005] These and other advantages of the invention will be apparent to those of ordinary skill in the art by reference to the following detailed description and the accompanying drawings.

Brief Description of Drawings

[0006] FIG. 1 is a conceptual representation of a content management architecture.

[0007] FIG. 2 is a conceptual representation of content redirection and content discovery in a network, in accordance with an embodiment of the present invention.

[0008] FIG. 3 is a conceptual representation of content redirection utilizing protocol redirect.

[0009] FIG. 4 is a conceptual representation of content redirection utilizing a dynamic helper file.

[0010] FIG. 5 through 9 illustrate different architectural options for content discovery, in accordance with an embodiment of the present invention.

Detailed Description

[0011] FIG. 1 is a conceptual representation of a content management architecture. Content can be distributed to clients from either origin servers, e.g. 161, 162, or from clusters of edge servers, e.g. 151, 152, ... 155. It is advantageous to have a content manager 110 which is able to selectively pre-populate some of the edge servers with content based on various factors, such as a negotiated service agreement, load, the particular service type, other policies, etc. In particular, for example, for high-quality streaming of media, it is desirable not to have all of the content populating all of the edge server clusters. In this type of environment it becomes crucial for a client to connect to an edge server that already has the content it is requesting. The present invention is directed to various mechanisms which facilitate effective content management.

[0012]

FIG. 2 is a conceptual representation of content redirection and content discovery in a network, in accordance with an embodiment of the present invention. The system, roughly speaking consists of two parts: a front end responsible for redirecting clients

and a back end that determines where the client should be redirected. A network 200 distributes content from multiple edge and origin servers to clients, e.g. client 280. The present invention is not limited to a particular content distribution architecture, although an advantageous architecture for the distribution of streaming content is described in co-pending commonly-assigned United States patent application, "Network Based Replay Portal," Serial No. 09/726,576, filed on December 1, 2000, which is incorporated by reference herein. At step 201, a media client 280 consults a local domain name system (DNS) server 211 with a URL for the particular content it seeks. The DNS server 211, at step 202, resolves the URL to a redirect server 220. The redirect server 220 queries a mapping service 230 responsible for determining where content is located in the system. The mapping service, as further described below, can comprise one or more mapping servers which receive updates from origin servers 261, ... 262 and from edge servers 251, ... 255. At step 204, the redirect server 220 redirects the client to an advantageous edge (or origin) server. At step 205, the client 280 requests the content from the media server.

[0013] *Redirection Mechanisms*

[0014] There are at least two places where the redirection can take place. For example, and without limitation, streaming media over an IP network typically involves first making a selection from an HTML web page. As a result of the selection, a helper file is downloaded and a media player application is started. The helper file contains a streaming URL (or URLs) which the player then uses to connect to the relevant media server by means of a streaming control protocol.

[0015] The first method of redirection involve dynamically generating the helper file with a URL (or URLs) containing the hostname (or IP address) of the edge server that the client should be connecting to.

[0016] The second method of redirection makes use of explicit redirection that is present in most streaming control protocols (including RTSP, HTTP and MMS (in version 7.0)). In this case the media client connects to a redirecting media server. This media server never serves up content but always redirects the client by means of the control protocol to the appropriate edge server.

[0017] Note the proposed solution also enables an additional benefit beyond finding the appropriate edge server: The requested content (object) is taken into account in making the redirection decision. The result of the redirection is a new URL which contains a new hostname (or more likely the IP address) of the selected edge server as well as the selected object. This allows the opportunity to also remap the requested object during the redirection process. The implication of this is that name of the object that a user selects and the name of the object it is actually served need not be the same. This property can be exploited in a number of ways. For example a CDN customer (i.e. a content provider) can use its own naming convention which gets mapped onto a naming convention within the CDN which is more efficient for the CDN provider. (E.g. a CDN provider might determine that the a piece of content provided by two different content providers are really the same piece of content (such as a movie for example) and therefore choose to exploit that within the CDN to have two different customer names map to the same piece of stored content.)

[0018] FIG. 3 and 4 illustrate the two different methods of redirection. FIG. 3 is a conceptual representation of media redirection utilizing a protocol-level redirect, specifically in the context of the RTSP protocol. At step 301, a media client 380 has a URL, e.g. "rtsp://sr.target25.com/clip.rm". At step 302, the domain name system resolves the domain name "sr.target25.com" to any of the streaming redirect servers 320. At step 303, the media client 280 connects to the chosen streaming redirect server 320 with the URL "rtsp://sr.target25.com/clip.rm". At step 304, the streaming redirect server 320 decides on an "appropriate" content server for this request, for example redirecting the client to an edge server 351 with the URL "rtsp://mec10.att.net/att/clip.rm". The IP address of the edge server 351 can be utilized to prevent a DNS lookup. Then, at step 305, the media client 380 connects to the media edge cluster to stream the desired content. The media edge cluster 351 may already have the content in a cache or may obtain the content from a media origin server 361.

[0019] Protocol-level redirection does not rely on any standardized naming convention. A URL such as "rtsp://sr_rtsp.target25.com/balloon.rm" can map to "rtsp://real.atticds.speedera.net/real.atticds/balloon.rm" or to "rtsp://live-att-icds.com/balloon.rm", for example with equal probability. Where a movie URL such as

"rtsp://sr.att.com/bb/the_messanger" is requested, the URL could be mapped to "rtsp://mec5.icds.att.net/movies?id=1011" assuming that the content is already on the server cluster "mec5" or could be mapped to "rtsp://mec5.icdns.att.net/bb/the_messanger?url=rtsp://mos3.icds.att.net/movies?id=1011" where the movie is still on a central server "mos3" but it is desired to stream the movie from server cluster "mec5".

[0020] FIG. 4 is a conceptual representation of content redirection utilizing a dynamic helper file. At step 401, a user utilizing a web client 490 selects a media clip from a web page on a web server 495. The web server 495, at step 402, dynamically generates a helper file (e.g., a ".ram" or ".asx" file) with the appropriate URL, e.g., "mms://mec10.att.net/clip.asf". At step 403, the web client 490 invokes the media player 485, which contacts the domain name system at step 404 to resolves "mec10.att.net". Alternatively, the URL could be expressed with an IP address to avoid the DNS lookup. Then, at step 405, the media player 485 connects to a media edge cluster 451 to stream the desired content. The media edge cluster 451 may already have the content in a cache or may obtain the content from a media origin server 461.

[0021] This method of redirection again does not rely on any standardized naming conventions and can redirect between different protocols. For example, a URL of "http://www.target25.com/cgi-bin/getasx.asx?url=balloon.asx" can be mapped so that it returns an asx file with "href=http://httpmode.att-icds.com/commercial/balloon.asf" or "href=mms://wm.atticds.speedera.net/wm.atticds/balloon.asf" with for example equal probability.

[0022]

Note that it is not possible to do content aware redirection with other known redirection methods, in particular, DNS-based redirection. The reason is that DNS resolution resolves a DNS hostname to an IP address. The content part of a URL is therefore not taken into account. It is in principle possible to somehow "encode" the content into the host name and therefore overload the DNS system to be content-aware. There would however be a number of (known) problems with such a solution:

- Load on the DNS system would dramatically increase as the system would be used in a way that it was not designed to be used.

- By design the decision making would be more course grained because of DNS caching/time-to-live, no knowledge of the client requesting IP address and the fact that many clients might be hidden behind a single DNS query.

With DNS based redirection schemes it is also impossible to perform any mapping of the requested object name as it is not available within the DNS query.

[0023] *Mapping Service*

[0024] The decision as to where a media client should be redirected to typically involves many inputs. This might include load in the network, load at edge servers, policy, locality of client to CDN nodes, network characteristics between client and CDN nodes etc. The key criteria we are concerned with in this writeup however is the content that is being requested and where this currently resides in the CDN. Or more generally how that content should be obtained. For example, for live content, the decision might be to serve it from a particular edge server because of its locality, and the way for the edge server to obtain the content might be to join an application level streaming distribution tree.

[0025] One way in which the backend could be realized is to have all components (origin servers, edge servers, streaming redirectors, etc) take part in a content exchange/content discovery process. As illustrated by FIG. 2, origin and edge servers advertise the fact that they have a particular piece of content to a mapping service 230. When a redirector 220 receives a request for content it in turn consults the mapping service 230 to find out if and where the content is available and redirects the media client appropriately.

[0026]

Thus, the primary function of the mapping service is to return the location of a piece of content, identified for example by a URN. In response to a query, the mapping service is responsible for returning a list of URLs specifying the servers on which the content may be found. A full query might look like this, in the context of a mapping service for television broadcast content:

```
<channel_name; brand; distributor; region; time; requestor_location>
```

The mapping service without limitation can be modeled as a huge, singletable

database containing the following fields:

```
Time, Channel Name, Brand, Distributor, Region, Portal
```

In this naive approach, there is an entry in the database for every piece of content on the content network, regardless of whether that piece of content is unique or replicated. In other words, if the same piece of content is mirrored in twenty locations, there will be twenty distinct records in this database.

[0027] The mapping service, in accordance with a preferred embodiment of the invention, can accept two types of message: queries and updates. Queries ask the mapping service to locate a piece of content, identified by its URN. Updates allow portals to add content to the mapping service, delete it, or change certain attributes pertaining to a piece of content. There are a number of implementation approaches for the mapping service, which are listed below. As shown below, the most important issues to consider when designing a mapping service are the size of the database, the search time and the quantity of traffic.

[0028] *1. Centralized Database.* In this solution, updates and queries are directed from the portals to a single, centralized mapping server, as illustrated by FIG. 5. To speed local searches, a "Local Mapping Server" (LMS) can optionally be placed in each neighborhood as an intermediary. The Local Mapping Server serves the purpose of aggregating local content, as well as performing all communication with the Central Mapping Server (otherwise referred to herein as a "Global Mapping Server" (GMS)). Conceptually, this is the most straightforward implementation of the mapping service. This solution has the disadvantage of being relatively vulnerable to database failure; if the server were to go down or become unreachable, a large portion of the functionality would become unavailable for searching.

[0029] *2. "Gnutella"net/Multicast Distributed Database.* In this configuration, illustrated by FIG. 6, each portal (or group of portals, represented by a Local Mapping Server) maintains a database of its own stored content. There are no update messages transmitted between these nodes. When one node wishes to query the database, a query is broadcast to all nodes (through a network flooding scheme, or through multicast.) Any nodes containing the content then respond directly to the node

initiating the query.

[0030] Unlike the singledatabase solution, this solution has the advantage of being relatively invulnerable to database failure; if one of the many nodes fails, only the content stored in that neighborhood becomes unavailable to the rest of the world. Database size is no longer an issue, as databases are maintained locally. Search times are lower for each local database, as these databases will be much smaller than a large amalgamated database (although the process of disseminating queries could potentially add a great deal to the response time.) Finally, if the queries are distributed along paths that resemble the network topology, it is likely that the earliest responses will come from those nodes "closest" to the querying node. The primary disadvantage to this solution is the large amount of query traffic, which would be broadcast to all clients; this flooding could potentially overwhelm the portals. Other problems include the lack of a "guarantee" of query failure---if no responses are received within a specified time period, querying nodes must simply assume that the search failed. By the same token, it is also possible that some searches will result in too many responses, temporarily overloading the querying node. Large amounts of broadcast query traffic would make this solution unworkable. As query messages are triggered by individual clients (viewers), it is possible that query traffic could be relatively large, possibly hundreds of thousands of queries per second. As this traffic is broadcast to all nodes, nodes might be unable to handle this enormous flood of queries. Although there might exist some mechanisms for reducing the quantity of query traffic (such as limiting the TTL of query requests) but these might not solve the problem.

[0031] *3. Multiple Partitioning (Partitioned by Channel/Time/Distributor/etc.) Database.* As illustrated by FIG. 7, this scheme improves on the scalability of the singledatabase approach, by partitioning the database into many smaller databases. Each of these subdatabases contains records associated with a given channel, time or distributor (for example.) When a portal wishes to perform a query, one of these subdatabases is chosen using information from the query URN. For instance, if the query URN contains the following information:

```
< ABC;WABC;comcast;new_jersey;10:30PM>
```

Then the querying node might address the query to the subdatabase server responsible for records relating to the channel "ABC", to the server responsible for the affiliate "WABC", or to the server responsible for distributor "comcast" (etc.) The actual address of the servers might be calculated simply by generating a domain name (e.g. ABC.prism.att.com, WABC.prism.att.com, comcast.prism.att.com.).

[0032] The decision on which database to contact would depend on how the sub-databases are partitioned. If information is partitioned according to only one parameter (channel, for instance), then the querying node's decision would be straightforward. If the partitioning was based on multiple parameters---if, for instance, there existed both a "comcast" database and an "ABC" database---then the querying node would have to choose which database to contact. If data is to be partitioned based on multiple parameters---for example, if we do have both an "ABC" database and a "comcast" database---then the contents of the "comcast" database would need to substantially overlap the contents of the "ABC" database. This could lead to a great deal of database replication, and also requires that updates received at one database ("comcast" for instance) would need to be propagated to all other relevant databases ("ABC" and "WABC".)

[0033] This solution is potentially an improvement on earlier models with regards to scaling in that it does reduce the quantity of query traffic handled by individual databases. However, it does not necessarily reduce the traffic in a uniform way. Whether we partition the databases by channel name or any other parameter, there is no guarantee that query traffic will be evenly distributed across the resulting sub-databases. Since there will be such a high degree of database replication under this scheme anyway, it might almost make more sense if we simply replicated the full database many times (as illustrated below), and asked querying nodes to pick a server at random from a list. This way, we could at least guarantee a certain amount of uniformity.

[0034]

4. Replicated Central Databases. One of the most desirable features of a distributed database is the ability to redistribute dynamically, based on changes in load and the number of records. If it is assumed that one of the challenges will lie in regulating the number of queries, then one could use the simple mechanism

mentioned in the previous paragraph: simply replicate the full database many times, and create a mechanism for distributing queries amongst the clones, as illustrated by FIG. 8. This approach would require a mechanism for propagating update traffic to all of the databases.

[0035] *5. Replicated Local Databases ("Gnutella"Net approach).* Another variation on the distribution of functionality would be for all portals within a neighborhood to constantly update each other about their stored content, as illustrated by FIG. 9. Each portal would therefore maintain a complete database of content available within its own neighborhood. To locate content that is not in the local neighborhood, this scheme designates a ``gateway" portal within each neighborhood to take part in a similar distribution of available content with gateway portals from other neighborhoods. The gateway portal would be responsible for aggregating the reports of content available within its own neighborhood, transmitting that information to all other neighborhoods, and collecting reports back from other neighborhoods. In this scheme, the gateway portal represents the neighborhood at the interneighborhood level and individual portals within the neighborhood are not visible from the outside during this part of the content discovery process.

[0036] In this scheme a query would be handled as follows: Upon receiving or initiating a query, a portal would first check its local database, which would contain a full record of all content to be found within the neighborhood. If the query item were not found, the portal would then send a request to the neighborhood's gateway portal, which would then look the item up in its much larger "global" database.

[0037]

This scheme could be repeated on larger scales, as well. Multiple domains can be linked in much the same way that different neighborhoods are linked in the above model. Again a designated node (this time from among the interneighborhood participants) would exchange information with peers in other domains. In this case, in addition to aggregating the reported content available within the domain, the designated portal would be responsible for implementing any policy filtering that might be required for exchange of content between domains. The advantage of this scheme is that the same basic scheme can be used at different levels in the hierarchy. Being completely distributed the scheme is potentially very robust against node

failures. For the common case this scheme localize queries (the rate of which is user dependent) at the expense of having to distribute updates (the rate of which can be largely controlled by the system) throughout the neighborhood.

[0038] The major disadvantage to this scheme is that it might be too complicated for a local scheme, while an interdomain mechanism might be better served by a query-based solution. While potentially more robust this scheme is a lot more complicated than a simple centralized approach. A major worry in this configuration is that the individual databases might get out of sync with each other. This could occur if a number of update messages are destroyed. Once this occurred, it would be a very difficult situation to recover from, and could result in important content become inaccessible.

[0039] Another possibility would be to sort the database according to the one parameter that is unique in streaming content: time. If records are sorted by time, they can be broken up across several databases and queries directed properly based on the time requested. If query traffic reached uncomfortably high levels on any one server, that server could dynamically repartition, dumping some of its more popular records onto another server. With this approach, a mechanism is required for mapping requests to the correct server, even as records were moved.

[0040] *Mapping Protocol*

[0041] The following is a description of an advantageous mapping protocol, which the inventors have named the "URI mapping protocol" (UMP). The purpose of UMP is to map one URI to another, e.g. it can map a URN to a URL as part of the content discovery mechanism described above. The protocol could however be used in any generic URI mapping application. Similar to HTTP, RTSP and SIP, the UMP protocol is text based. It also attempts to reuse many of the header fields defined by these protocols.

[0042]

The following provides a framework in which a more detailed protocol specification can be developed by one of ordinary skill in the art depending on the requirements and understanding of the problem space. In particular, in a distributed realization of the protocol, it will be necessary to add messages (and other support) to

build the distribution network. There description does not specifically address any security or authentication issues, although known ideas from HTTP can clearly be reused.

[0043] This protocol can run over either a reliable transport protocol (e.g. TCP) or an unrealiable transport protocol (e.g. UDP). Following the example of RTSP and SIP a retransmission scheme is used only in the case where an unreliable transport is used. Similarly, since IP level fragmentation is undesireable, it should be required (like SIP) that UDP datagrams fit in one MTU if it is known or in 1500 bytes which is considered a reasonable default MTU size. Again following the SIP example, more compact representation of encodings can be used to reduce the per message length. For UMP over UDP, it is required that a response that would not fit in an MTU be truncated and sent with an appropriate error indication. Even though it is not the complete intended message, the truncated part of the message should be a properly formatted message. For example, a query that resulted in 20 mappings should return 15 complete mappings, assuming that is all that can fit in the MTU, rather than 15.5 mappings, with an error indication that all results could not be returned. Alternatively, a response to a UDP received query can be sent back over TCP, if allowed by the client.

[0044] The default operation for UMP is a "request-response" type interaction where every request message is acknowledged by the recipient by means of a response message. Pipelining of requests are allowed and in this case the order of responses do not necessarily have to match the order or requests. Receipt of the response message by the requestor terminates this transaction (indicated by the transaction ID in the message pair). In the absense of other mechanisms, failure to receive a response after a reasonable timeout can be used by UMP to trigger retransmission in the case where UMP runs over UDP. An alternative is to make use of some reliable transport layer between UMP and UDP. In UMP there are two exceptions to this default one-request-one-response form of interaction:

- In a distributed query operation, the first recipient of the query might be able to resolve it (i.e. produce a response message), but it will also pass the request on to its neighbors in the distribution network. These neighbors in turn might be capable of resolving the query (and sending it back towards the recipient) and will similarly pass on the request to their neighbors. In this mode of

operation, the requestor (and nodes along the distribution network) should therefore be capable of processing multiple responses for each request. To accomodate UDP based implementations which might rely on the response for its retransmission scheme, a node should send back a response to the previous hop if it is forwarding the request to another node. This should be done even if the node in question was unable to resolve the query. If the node was able to resolve the query, the provisional response should contain these responses. Such a provisional response should indicate that it is not a final answer. The default case would be to use the provisional response for UDP but not for TCP, but the client is allowed to overide the default.

- In a distributed update operation, e.g. by using native multicast, the requestor can indicate that it does not require an acknowledgement on its request. Since reliability can not be guarenteed in this mode of operation it is assumed that consistency will be provided by some other means, e.g. periodic refreshes and time-to-live values associated with mapping entries, or by making use of reliable multicast.

[0045] It is also advantageous to provide an OPTIONS message pair as well which will establish connectivity between communicating entities, telling the other end what type of node it is (e.g. client/server versus distributed operation transport protocols it can use, etc.).

[0046] The use of one URI per Query-Request and one URI mapping (e.g. URN to URL) per Update-Request simplifies the protocol. Use of multiple URI in Query-Request does not appear that compelling and it does complicate processing (e.g. what happens when a local mapping server can resolve some of the URIs locally but need to contact another entity to resolve the rest). For Update-Request the usefulness of multiple URIs seems more compelling. However, in this case pipelining the updates in one transport level message will work well enough albeit more inefficient. If this prove too limiting in practice, it is always possible to extend the protocol later on to accomodate multiple entries.

[0047]

The following is a specification for the query messages, in accordance with a preferred emb

```
Query-Request: ------------- R QUERY <query-URI (URN)> UMP/1.0 R Via: <list
```

```
Query-Response: -------------- R UMP/1.0 Status-Code Reason-Phrase R Via: <l
```

Note that "FromURI" has to be in response so that a querier know what the mapping is even if th

[0048] The following is a specification for the update messages, in accordance with a preferred em

```
Update-Request: -------------- R UPDATE <update-URI (URN)> UMP/1.0 R Via: <l
```

```
Update-Response: -------------- R UMP/1.0 Status-Code Reason-Phrase R Via: <
```

Note that the "update-URI" should be fully qualified with no range specification.

[0049] The "TransactionID" should be an opaque string of at least 8 octets. The "TransactionID" must be unique for every unique request. It can, without limitation, have a random component to it and/or can be globally unique when considered together with the sender's hostname.

[0050] It is also advantageous to add the following new status code:

```
New Status-Code Reason-Phrases ============================== 230 Provisiona
```

[0051] The following are examples of the use of the above protocol for three scenarios: (1) where there is a centralized local mapping server (LMS) with a centralized global mapping server (GMS); (2) Distributed (replicated) LMS / distributed (replicated) GMS; and (3) Distributed (partitioned) LMS / distributed (partitioned) GMS.

[0052] *Centralized LMS with centralized GMS.* Consider where there is an initial update from a port

```
Request (from portal11 to lms234): ------------------------------------ UPDAT
```

The LMS will then send a similar update to the GMS (it might be immediate or it might wait for a

```
Request (from lms234 to gms2): ----------------------------- UPDATE stv:<abc
```

[0053] Suppose a client request for stv:<abc;;;> is received by a portal in the same neigborhood as

```
Request (from portal12 to lms234): --------------------------------- QUERY s
```

[0054] Suppose a client request for stv:<abc;;;> is received by a portal in a different neighborhood

```
Request (from portal31 to lms456): -------------------------------- QUERY st
```

In this example, RTSP URLs are never visible outside the local neighborhood and a query therefo

[0055] *2. Distributed (replicated) LMS/distributed (replicated) GMS (gnutella-like updates).* Assume

```
Request (from portal11 to each of its neighbors in the network): -----------
```

At this point the update is done as far as portal11 is concerned. Each of the neighbors that has

```
Request (from portal13 to each of its neighbors in the network): -----------
```

This process continues untill all portals in the neighborhood have received the update. The des

```
Location: ump://portal15.att.net
```

This will allow the same hiding of the details of any particular neighborhood as in the previous

[0056] Any query received for stv:<abc;;;> in portal11's neighborhood will be resolved locally by th

```
Request (from portal31 to portal39): -------------------------------- QUERY
```

As before portal31 will then proceed to query portal15:

```
Request (from portal31 to portal15) -------------------------------- QUERY s
```

What is interesting in this approach is how portals know at what level(s) of a hierarchy they sho

[0057] *3. Distributed (partitioned) LMS/distributed (partitioned) GMS.* In this approach there are no updates. Instead queries are broadcast gnutella style through an overlay network. We assume that similar to the previous section a distribution network has already been established and that it is organized into neighborhoods. Designated nodes in each neighborhood form a higher level group by similarly having a distribution network amongst themselves. The "RegionTTL" field is used to determine the scope of a query.

[0058] As before, assume that portal11 carries stv:<abc;wabc;comcast;summit.nj>. A client reques

portal13 to each of its neighbors: -------------------------------- QUERY s

Negative acknowledgements are not relayed back to the original requestor. A designated node

[0059] A query from a different neighborhood will be handled in similar fashion and assuming the scope of the query is higher than 1 will eventually via the higher level network end up in the neighborhood where portal11 resides. portal11 will respond in exactly the same way as above so that the requesting portal will end up with the URL for portal11. (In this case there is no LMS in the loop.)

[0060] Note that in the case where multiple responses are possible, an implementation (at the initial requestor side) must be capable of receiving (and discarding) reponses (long) after it might have cleaned up any state associated with the request. Other solutions to this problem involve the first upstream node maintaining state so the responses can be merged before being sent to the requestor and subsequent responses discarded.

[0061] It should be noted that it would be advantageous to include something in the protocol to indicate which protocol to use for interaction. It might be preferable to have a PortalID as additional information (e.g. this could simplify the index operation when an update is received). It may be preferable to ensure that redirects (e.g. from LMS to GMS back to LMS to GMS, etc.) will not occur and that when they do occur that the situation will not be disastrous. Also, the via fields will prevent loops from forming but will not prevent messages from being processed many times by the same node in the distributed cases. This might be addressed by requiring the distribution network to be a tree or by having nodes cache the transaction ids of recently seen messages (as in Gnutella). Finally, it should be noted that it is possible to do away with query response messages and only have updates, which can be either solicited or unsolicited.

[0062]

The foregoing Detailed Description is to be understood as being in every respect illustrative and exemplary, but not restrictive, and the scope of the invention disclosed herein is not to be determined from the Detailed Description, but rather from the

claims as interpreted according to the full breadth permitted by the patent laws. It is to be understood that the embodiments shown and described herein are only illustrative of the principles of the present invention and that various modifications may be implemented by those skilled in the art without departing from the scope and spirit of the invention. For example, the detailed description has been presented in the context of particular protocols such as RTSP and HTTP; however, the principles of the present invention could be extended to other protocols and content distribution architectures. Such an extension could be readily implemented by one of ordinary skill in the art given the above disclosure.